Criterion-Based Grading, Agile Goal Setting, and Course (Un)Completion Strategies

Ihantola, Petri

Springer
2019

Ihantola , P , Isohanni , E , Heino , P & Mikkonen , T 2019 , Criterion-Based Grading, Agile Goal Setting, and Course (Un)Completion Strategies . in D Parsons & K MacCallum (eds) , Agile and Lean Concepts for Teaching and Learning : Bringing Methodologies from Industry to the Classroom . Springer , Singapore , pp. 207-230 . https://doi.org/10.1007/978-981-13-2751-3_11

http://hdl.handle.net/10138/320591
https://doi.org/10.1007/978-981-13-2751-3_11

unspecified
acceptedVersion

Criterion-based grading, agile goal setting, and course (un)completion strategies

Petri Ihantola[1], Essi Isohanni[2], Pietari Heino[3], and Tommi Mikkonen[4]

[1,4] University of Helsinki, Helsinki, Finland
[2,3] Tampere University of Technology, Tampere, Finland
[1] petri.ihantola@helsinki.fi

Abstract. When teaching large groups of students with heterogeneous backgrounds and different learning goals, it is essential to personalize the learning experience. In this chapter, we describe how we have implemented this in a university-wide introductory programming course. Each student sets a personal target grade, i.e., the grade they aim at, based on how deep an understanding of programming they need (depending on their major subject, etc.) and on how much effort they are willing to invest in the course. To enable such setup, course assignments are divided into different levels and the grading directs the students in choosing which assignments to work on to meet the goals they have set. Furthermore, the students can change their target grade during the course in an agile manner.

Keywords: Criterion-based grading; automated assessment; CS1; student strategies; agile goal setting

1 Introduction

The constructivist learning theories propose that a learner constructs their own comprehension of the subject through their prior knowledge (Illeris, 2002). These theories emphasize that learning is an individual process that reflects the back- ground of the learner. Therefore, it is essential to let the learner personalize the learning process by choosing learning materials according to personal preference.

In self-directed learning – typical in adult education – the learner also takes the initiative to formulate and pursue learning goals (Merriam, 2001). Unfortunately, this self-imposed setting of goals is often poorly supported. As Lister and Leaney (2003) state:

> " In the traditional [...] approach to grading, all students in a CS1 class attempt the same programming tasks, and those attempts are graded "to a curve". The danger is that such tasks are aimed at a hypothetical average student. Weaker students can do little of these tasks, and learn little. Meanwhile, these tasks do not stretch the stronger students, so they too are denied an opportunity to learn."

In the same article, Lister and Leaney propose a criterion referenced grading scheme, where the students do different assignments, according to their abilities. Moreover, the assignments are designed to match the cognitive domains of Bloom's taxonomy (Bloom et al., 1956), a classification of levels of intellectual behavior important in learning (Seddon, 1978). While the taxonomy consists of three hierarchical models – cognitive, affective and sensory domains – the cognitive part has been the primary focus of most traditional education. In particular, it has been commonly used as basis for structuring curriculum learning goals, assessments, and activities (Fuller et al., 2007).

In this chapter, we describe how we have implemented the criterion-referenced grading scheme in the context of a university-wide introductory programming course. In addition, we introduce an agile course concept: We explicitly ask each student to choose their learning goals, which define the assignments they should complete. In connection to agile software development, the analogy is to allow the team to decide which features to pick from the product backlog. Here, the students decide how much work they are willing to invest in learning a certain topic in the course, and pick assignments with corresponding complexity. The approach was designed to support both struggling (Ahadi, Lister, Haapala, & Vihavainen, 2015) and over performing (Carter et al., 2010, 2011) students by providing a personal, agile learning experience despite extremely large teaching groups.

Struggling and over-performance are often related to a mismatch between prior skills and learning goals. We have divided all our programming assignments into four categories, with increasing complexity. The way this is done resembles grouping the levels of Bloom (Lister & Leaney, 2003; Johnson, Gaspar, Boyer, Bennett, & Armitage, 2012). However, as applying Bloom's taxonomy consistently in Computer Science (CS) education can be very challenging (John- son & Fuller, 2006; Thompson, Luxton-Reilly, Whalley, Hu, & Robbins, 2008), our approach is more practically oriented: Skills learned from the higher category assignments are prerequisites of a future course only if a student is planning to take programming as their major or minor. Moreover, assignments from the lower categories may be too simple for students with prior knowledge. Thus, they are optional for students aiming at higher grades.

Our main tool for organizing the course in an agile way is the grading rules of the course. In our setup, the final grade is based on solving automatically graded programming assignments throughout the course. The approach is applicable also in other contexts, however. We describe the course to give an overall understanding and to reflect our results. In addition to explaining the details of how we have implemented agile into education, our main objective is to seek understanding regarding how students behave in the setup.

The rest of the chapter is structured as follows. First, in Section 2, we introduce the agile course setting, which forms the background of this paper. Then, in Section 3, we describe the research questions related to students' behavior and the related methodology. Next, in Section 4, we introduce our results, and in Section 5, we provide an extended discussion regarding our main observations. Finally, in Section 6 we draw some conclusions.

2 Agile Course Setting

In order to describe how the agile goal setting has been implemented in our introductory programming course, we first describe why the course is needed in our university and who takes it (Section 2.1). Next, we define the teaching methods (Section 2.2), and the grading scheme enabling the individual learning paths (Section 2.3). Finally, the idea of the grading scheme is illustrated with examples of different learning goals (Section 2.4).

2.1 Versatile Needs

The overarching learning goal of the Introduction to Programming course in Tampere University of Technology, Finland, is to learn to write small programs on one's own. We use Python as the programming language, and in addition to basic computer usage skills, there are no other prerequisites for the course.

The course is obligatory for almost all the students in the whole university, more than 1000 students every year. Consequently, large and heterogeneous student groups are in-

cluded – in addition to computer science students, also for instance electrical engineering, automation engineering, mechanical engineering, and even material science students take the course. The background for this decision is that in the modern world all graduates need to know at least the basics of programming to better understand the use of computer applications in their own fields.

While computer science students build almost all of their professional skills on top of their ability to program, the students in many other fields just need to understand what programming is. In addition, students' previous programming skills vary greatly: most of the students start with no previous programming knowledge, but the teaching group also includes students who have been programming in high school and in their free time.

The diversity of the teaching group is illustrated in Figure 1. Students in Group A are majoring in some other field than computer science. Consequently, they only need elementary programming skills. Group B consists of students who start with no prior programming experience and need to cover all the topics of the course. They need to work really hard in this course. Group C consists of students that know the basics prior to the course. Thus, they cope even without special attention from the teacher and are most often neglected in large teaching groups. In our case, all of these students are attending the same course, and hence our goal is to meet the needs of all these student groups in our pedagogical design.

Another solution would be that all the different curricula that need programming would be free to create their own courses with small number of students participating in each of them. However, our solution where all students attend the same course ensures flexible possibilities of changing study plans for students. There are many students who do not know what programming is before they at- tend the course. In our course they can decide that they want to cover the topics more thoroughly and proceed to further programming courses. In addition, in comparison to going for different courses, this model allows fine-tuning the goals in accordance to students' views, not only on individual course definitions.

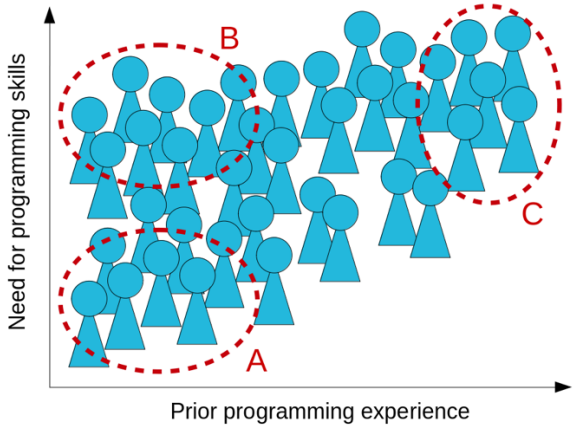

Fig. 1. Diverse student population, with special target groups A, B, and C identified.

2.2 Practical arrangements

All the course material is available online, delivered by using the A+ course platform (Karavirta, Ihantola, & Koskinen, 2013). The material contains an e-book where all the theory is explained and embeds automatically assessed programming assignments with immediate feedback. Each assignment can be submitted for evaluation at most ten times. The automated approach is widely applied in programming education (Carter et al., 2003; Douce, Livingstone, & Orwell, 2005).

In addition to automated feedback, the students also receive feedback from teaching assistants for selected assignments. These selected assignments test the core ideas of programming thus giving the students first hand feedback on whether they've actually understood the problems at hand or if there's something to improve on and pay attention to in the future.

The course spans over two teaching periods (in total 14 weeks), and it is 5 ECTS credits in size. A number of assignments are to be completed weekly. We use the flipped classroom method (Bishop & Verleger, 2013), where the students first complete the programming assignments and then attend the class to discuss different ways of solving the assignments and the problems they faced. Students work self-guided and ask for help from the teaching staff if necessary. To this end, we use a multipurpose computer classroom, where the students are allowed to work whenever they want. In addition, there are teaching assistant hours in the multipurpose classroom at least 20 hours a week.

At the end of the course students take an electronic exam which is like a skills demonstration where the student shows under controlled conditions that he/she can complete a small programming assignment independently. The exam is not supposed to be difficult. Essentially it is a programming assignment of the same style as provided during the course.

Our course design is learner-centered. The students work according to their personal weekly schedules and the teaching assistants are available upon request in the multipurpose classroom at least during peak times. This way of working requires a lot of self-discipline from our students. On the other hand, it is very flexible. We also highlight that working in their own schedule does not mean working alone. They are encouraged to work in pairs and also to discuss their problems both with the teaching assistants and in the lectures.

The course design and all the practical arrangements follow the principle of constructive alignment (Biggs & Tang, 2007) in which teaching and assessment methods are chosen to meet predetermined learning goals. As the learning goal is to learn to implement small programs on one's own, that is exactly what the student does every week during the course in all the assignments and in the exam.

2.3 Grading

Our course is graded on a scale from zero (failed) to five (the best). The final grade is defined by the completed assignments. In the A+ course platform, there are over 120 programming assignments, which we divided into four categories:

- **A elementary:** small assignments, where the student mostly just repeats something that was exemplified in the materials.
- **B basic:** further small assignments, which mainly concentrate on one new topic, but are more difficult than elementary assignments, because the solution is not directly in the materials.
- **C applied:** typically, the student has to combine knowledge related to more than one topic. Some of these assignments are so large that we call them projects, despite the students working on them only for a few hours.
- **D advanced:** these assignments require the student to get familiar with materials outside the core content of the course or are in some other way more difficult than assignments in the other categories.

Points are allocated to assignments in accordance to how laborious they are. The largest ones constitute approximately 100 points and the smallest 10 points. There is a deadline every week, with an optional, discouraged extension – if an assignment is submitted

during the extension, one can only receive 70% of the associated points. Even during the extension, it is possible to meet the teaching assistants in the class. This is important, because sometimes automated assessment causes problems in the submission phase. Assignments are associated with grading rules. Table 1 presents a slightly simplified version of the grading rules table. To achieve grade 3, the student should collect 400 points from the elementary, 900 points from the basic, and 400 points from the applied assignments, and pass the extended exam. Both exams are only graded as pass/fail. The verbal explanation for grade 3 is "good" in our university. Thus, the requirement is that the student achieving this grade knows the core content of the course well enough to apply it in practice (applied assignments, level C). Students targeting grades higher than 3 also need to accomplish advanced assignments. If the student has set the target lower than grade 3, he/she is allowed to pass the course with less work. However, a grade of 3 is a prerequisite for the more advanced programming courses.

Table 1. The grading rules table lists what are the requirements for each grade.

Grade	A-points (elementary)	B-points (basic)	C-points (applied)	D-points (advanced)	Exam2
1	600	700	-	-	basic
2	600	800	200	-	basic
3	400	900	400	-	extended
4	200	900	400	200	extended
5	-	700	400	500	extended

As different students achieve different skill levels, there are two versions of the exam, basic and extended. The grading rules (Table 1) also define which exam the student has to take. The extended exam is not necessarily any harder than the basic one, but it covers a wider selection of topics. The role of the exam is only to double-check that students did their assignments independently enough to reach the learning objectives. Therefore, the exam is graded pass/fail and failing it will cause failing the course.

It is possible to patch up missing points by completing assignments from the higher categories. The purpose of this rule is to enforce that missing a single deadline is never fatal. If you miss deadlines, the first consequence is that it is impossible to gain the grade 5. By missing more deadlines, the grade 4 will also be unreachable, and so on. To fail the course completely, the state of affairs has to be such that the student has repeatedly missed deadlines.

On the course platform, the student is presented with progress bars showing the number of collected points in each category. It is easy to follow how the points accumulate. In addition, the fully completed assignment scores are shown in green-colored circles and partly completed in yellow. Figure 2 shows the students view of the course platform. On the right-hand side, the student sees the accumulated points from different assignment categories and in the middle the progress of different rounds; the first four rounds (weeks) that have already been closed are collapsed while the last one is open.

2.4 Individual Learning Goals and Learning Paths

As already pointed out, the overarching learning goal of the course is that the student is able to implement small programs independently. However, in the spirit of the criterion-referenced grading (Lister & Leaney, 2003), with such a diverse student group in question, the students' individual learning goals may vary greatly.

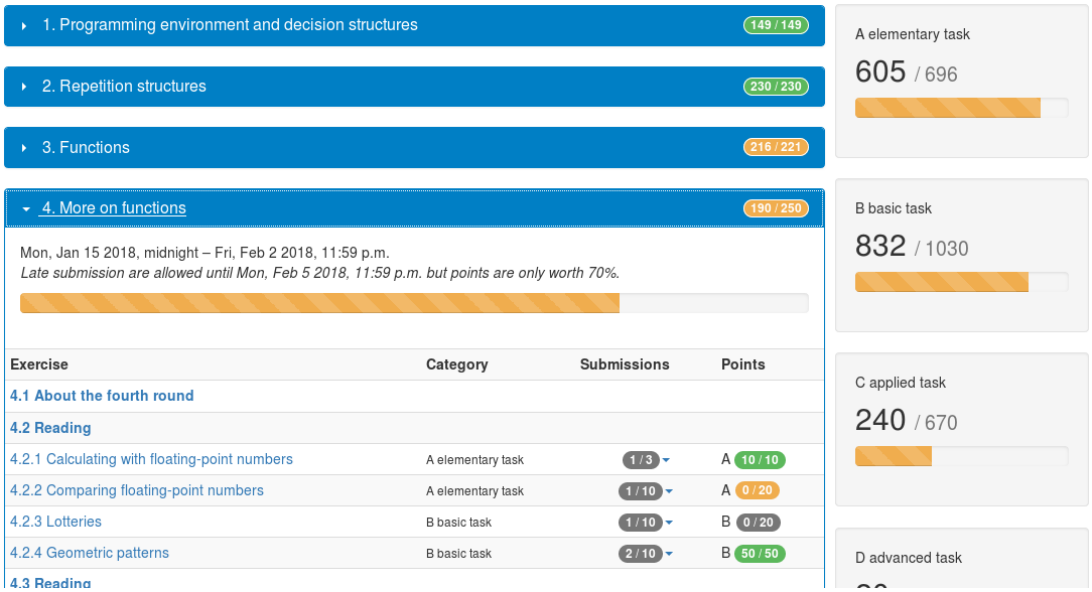

Fig. 2. Screen shot of the course platform showing progress bars and completion status of different assignments

All the students passing the course are required to meet the learning goal of implementing small programs independently. This means all students cover the basic concepts of Python—i.e. functions, lists, dicts, for instance—in some way. In levels A and B, the learning goal is just to get familiar with all these concepts. In level C, the students need to be able to apply this knowledge in more difficult situations. For example, in addition to knowing the data structures list and dict, they need to be able to combine the data structures, i.e. implement a program where the items of a list are dicts, for instance. In level D, the assumption is that the student will go further with programming studies and thus needs to learn how programmers search for information in work life. Therefore, some of the D assignments require searching for information in the Python documentation.

In contrast to traditional grading, here the grade that the student receives does not describe how well the student can implement given tasks but how widely he/she has covered the topics handled in the course.

Setting individual learning goals does not mean that the students who only cover assignments in levels A and B cover all the materials but do it somehow worse than other students. It means that they have identified they do not need to cover all the materials. They can still do all their work thoroughly. For example, for a student of some other discipline than computing it is enough that he/she is able to apply the information provided in the course materials and there is no need to learn to use the Python documentation.

Figure 3 presents two simplified illustrations regarding how the course can proceed. The time-axis of the illustrations runs horizontally from left to right. There are a number of assignments on different levels each week. In the beginning of the course there are mainly elementary and basic assignments and, in the end, applied and advanced. Figure 3a illustrates the course completion of a student who has set a very low target grade and only worked on level A and B assignments. Figure 3b illustrates the course completion of a student who has had prior programming experience already when starting the course and thus has skipped all the A-level assignments. When comparing these two accomplishments, we can almost say that these two students have almost taken a different course because their learning paths differ so much from each other despite them taking the same course.

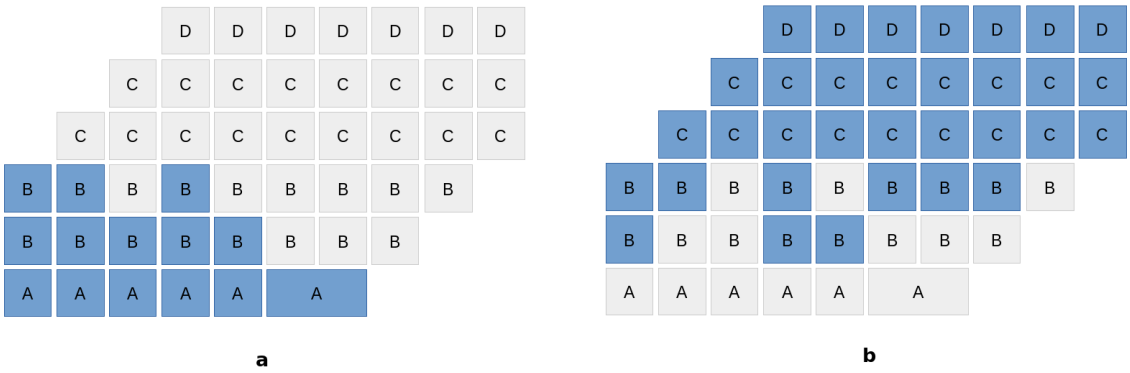

Fig. 3. Illustrating the differentiation of learning paths in our course setting.

This way of implementing differentiation of learning paths allows us to pay special attention to the student group C (Figure 1), which is often neglected simply because other student groups need more attention. As no student in the course is supposed to complete all assignments available on the course platform, some of the advanced ones can be so difficult that they also challenge the students with previous programming experience. Some of the advanced assignments can also cover topics that are not important for all the students.

The students — especially those with no prior programming experience — are encouraged to complete as many assignments as possible in the first weeks of the course. If they don't know anything about the course content, it is almost impossible to set the targets. At the end of the third week of the course, they have already seen assignments at all difficulty levels and know a little better so it is possible to set the target grades. At this point, we inquire what their targets are for this course. Despite their answers, nothing prevents the students from changing their targets later on, however. For example, many of the advanced assignments are "deep diving" into a specific topic. Skipping advanced assignments in the first half of the course does not prevent the student from completing assignments from the advanced category in the latter half of the course. Naturally even easier is to stop working on the assignments from the higher levels.

3 Research Questions and Methodology

In the previous section we presented the agile grading scheme and how we assumed students would apply it. In this section we describe the research setup aiming at understanding how students really use the setup.

3.1 Research Questions

In addition to describing our implementation of the agile course setup, the main objective of this study is to understand how students utilize the agile course setup defined in the previous section. The related research questions are:

1. How are students' self-reported target grades (i.e. individual learning goals) in the beginning of the course related to the final learning outcomes?
2. What kinds of behavioral patterns can be detected among students in the agile course setup?

Studying these research questions helps us to understand the course setup from the perspective of the diverse student population (Figure 1) attending our course.

3.2 Data

To answer the research questions, we collected data in two course implementations during the academic year 2016-2017. The CS students were mainly attending the first implementation (Autumn 2016). Students from the other disciplines attended both implementations (Autumn 2016 and Spring 2017). We had access log data of all the submissions from the course platform, grade targets set after the third week and final grades.

When comparing the grades in RQ1, we analyzed the data of all the students who were active in the course platform during any two weeks of the course, or who answered the question about the grade target. This is because in our context it's quite common that students sign up to the course platform just to see what the course looks like and may then immediately drop out from the course. The selected inclusion criteria resulted in 831 students; 527 in the 2016 course version from where 482 (91%) answered the grade target question, and 304 in the 2016 course version from where 267 (88%) answered the question. When answering the RQ2, we focused on the subset of the students who had answered the question about the grade target.

3.3 Methods

To answer RQ1, addressing the effect of defining an individual learning goal, we used Wilcoxon signed-rank test to compare the grade distributions of students who set their grade targets and students who did not. In addition, for students who set a target, we examined the predictive power of a linear regression model to estimate the final grade based on the target.

In RQ2, we applied visual data analytics to identify different behavioral pat- terns among the students taking the course. As defined by Keim (2002): *"The visual data exploration process can be seen a hypothesis generation process: The visualizations of the data allow the user to gain insight into the data and come up with new hypotheses"*. This kind of manual exploration of visualizations is typical in learning analytics and educational data mining (Mazza & Milani, 2005; Romero & Ventura, 2010).

Two teachers of the course who had knowledge on the content of the assignments looked at the visualizations and identified students with similar characteristics in their course completion paths. Examples of visualizations will be provided later on in this paper in Section 4.2. Patterns were identified and the data was then re-analyzed in order to confirm visualizations can be divided into these categories. Although visualizations of all study paths were viewed for identifying archetypal course completion patterns, the exact frequencies of the different learning strategies were not calculated.

4 Results

In this section, we describe the phenomena discovered in the data using both the visualizations and the additional data related to the course context. Section 4.1 is related to RQ1 and Section 4.2 to RQ2.

4.1 Comparison of Grades and Targets

Table 2 presents the distributions of the final grades in both course versions. Presentation separates students with and without grade target set. As expected, grades in the autumn 2016 course version (with CS-majors) are higher than in spring 2017 when participants were mostly non-majors.

We calculated Wilcoxon signed rank test with continuity correction to compare grades of students who answered the questions about the goal grade and students who did not. Non-parametric method was selected because of the relatively small number, and skewed distribution of the students who did not set their target grades (see Table 2). Students who answered the question performed significantly better with $Z=5232$, $p< .000$ and $Z=3683$, $p=0.004$, respectively in 2016 and 2017 course versions. We conclude that merely setting a grade target (i.e., an individual learning goal) seems to play an important role in performance. The grade difference in terms of median grades in 2016 was one and in 2017 two grades. Defining a goal seems to be especially important in passing the course. As illustrated in Table 2, dropout rates in 2016 were 17.4% and 42.2%, respectively for students with and without an individual goal. The same stats in 2017 were 40.1% and 70.3%.

Table 2. The grade distribution of students in both the course implementations.

Grade	2016		2017	
	goal set	no goal	goal set	no goal
0	84 (17.4%)	19 (42.2%)	107 (40.1%)	26 (70.3%)
1	58 (12.0%)	13 (28.9%)	44 (16.5%)	4 (10.8%)
2	79 (16.4%)	7 (15.6%)	30 (11.2%)	-
3	51 (10.6%)	5 (11.1%)	12 (4.5%)	1 (2.7%)
4	99 (20.5%)	-	47 (17.6%)	4 (10.8%)
5	111 (23.0)	1 (2.2)	27 (10.15)	5 (5.4%)

For students who set their targets, we constructed simple linear regression models to predict the final grade based on the self-reported grade target. Both course versions resulted to a significant model with adjusted R^2 of 0.277 ($F(1,481)=185.71$, $p<0.000$) and 0.18 ($F(1,265)=58.89$, $p<0.000$), respectively for the 2016 and 2017 course versions. The exact models and related illustrations are provided in Fig. 4. Negative offset and slope less than 1 in both models indicate that estimates are often optimistic. Means of the target grades among the students who defined that were 3.7 and 3.4, whereas means of the actual grades were 2.7 and 1.7, respectively for the 2016 and 2017 course versions. Moreover, we found that it is almost impossible to exceed your own expectations if the grade target is low (i.e., 1 or 2). Only one student who set the target grade at 1 or 2 achieved a better grade. Among the students who set a reasonable target grade, e.g. 3 or 4, there were more of those who exceeded their expectations (see discussion on casual well performers in Section 4.2).

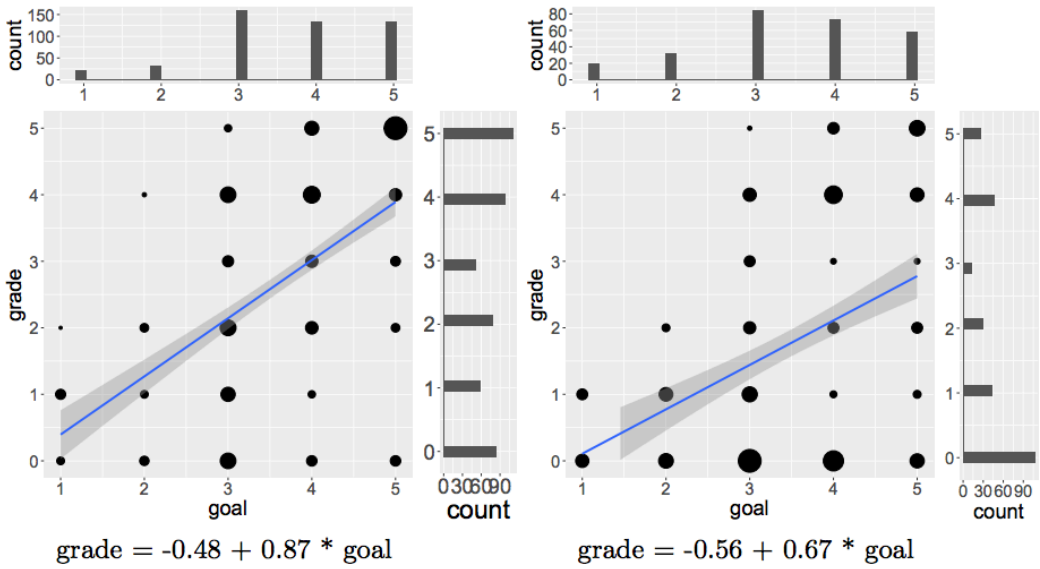

Fig. 4. Self-reported grade goal against the final grade and the regression model to predict the latter based on the target grade. Autumn 2016 course version (with CS students) is on the left. Spring 2017 course version with only few CS students attending is on the right. The difference in student background is clearly visible in grade goals as well as eventual grades.

4.2 Students' strategies

To answer RQ2 (what kinds of behavioral patterns can be detected among students in the agile course setup) we started by looking at students who got the best grades. The rationale of this was to identify different strategies that that lead to similar outcomes. After manual inspection of the data, we identified the following archetypes of students:

1. **Perfectionists**, who set their goal high and do practically all the available assignments,
2. **Opportunists**, who set their goal high, and did not do the optional (i.e. easier) assignments.
3. **Casual well-performers**, who did not set their goals too high but who still got a good grade at the end.

The exact frequencies of the patterns are not calculated. However, in comparison to perfectionists, the number of opportunists is small. Among casual well-performers there is a continuous spectrum from perfectionists to opportunists, again dominated by perfectionists – or at least students who do more voluntary A-exercises. This is not surprising, as casual well-performers started with the assumption they will actually need some A-points (i.e., lower grade target). In addition to high performers, dropouts turned out as interesting subgroup and the following patterns were recognized among them:

4. **Dropouts by missing assignments**: Students who after some point were not able to pass the course or get the desired grade and stopped immediately after that.
5. **Dropouts by missing skills**: Students who completed enough assignments and even participated in the exam (multiple times) without passing it.
6. **Dropouts by failing personal expectations**: Students who gained enough points to pass the course but not enough to meet the requirements of the target grade failed the course because they did not show up in the exam.

In addition to the above observations, two generic behavioral patterns that extend over the whole student population, almost disregarding their eventual grades, are:

7. **Goal-driven students** will stop all their activities immediately after they have gained enough points for their target grade.
8. **Compensating students** who have missed a significant number of assignments but who then end up compensating the missing points by doing more demanding assignments.

In the following we study these groups in more detail by using visualizations of their course completion paths.

Group 1: Perfectionists: This group of students did not care for the point requirements much. They completed all or almost all the assignments despite they had already gained enough of points for grade 5. There were more of these students in the first course implementation that the CS students took.

Figure 5 illustrates a typical course completion path for a student in this group. It is typical for enthusiastic learners to be interested in completing challenging assignments like the ones we had in level D. However, this student has also completed almost all A-assignments. It is especially noteworthy, that the A- and B-assignments in week 9 were all clearly instructed to be aimed for students who did not cover this topic in the earlier week. The students who had covered it, for instance the student presented in Figure 5, were instructed to proceed directly to the C-assignment of this week. Still this student has completed them all, exactly like all the other assignments too.

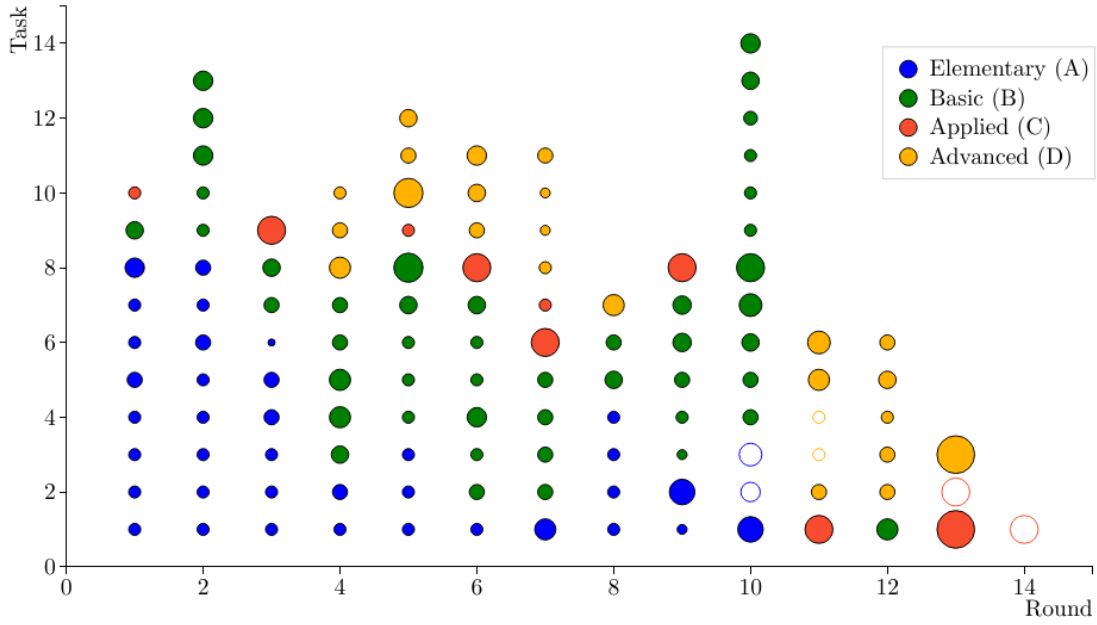

Fig. 5. Perfectionist. Student who targeted grade 5 and achieved it. He completed practically all the assignments in the whole course. His only non-completed assignments are so called ”extra assignments” that were open after the end of the course and assignments he was not required to do.

For this student, the course platform contains lines of green-colored circles and full progress bars. We assume that this was one of the motivational reasons for completing so many assignments that did not contribute to learning. This is why we have named the group perfectionist

Group 2: Opportunists: Although it was not usual, there were also students who took the advantage of not having to complete the A-level assignments since they were targeting grade 5 and had prior programming experience. Figure 6 illustrates one such completion path. In some of the weeks this student has completed some of the easiest assignments probably to check what the new materials are about. However, he has mainly concentrated on completing the applied and advanced assignments, which is more meaningful for a student who already has the basic knowledge. The week 10 where he has completed multiple B-assignments was about object-oriented programming, which was probably a new topic for this student.

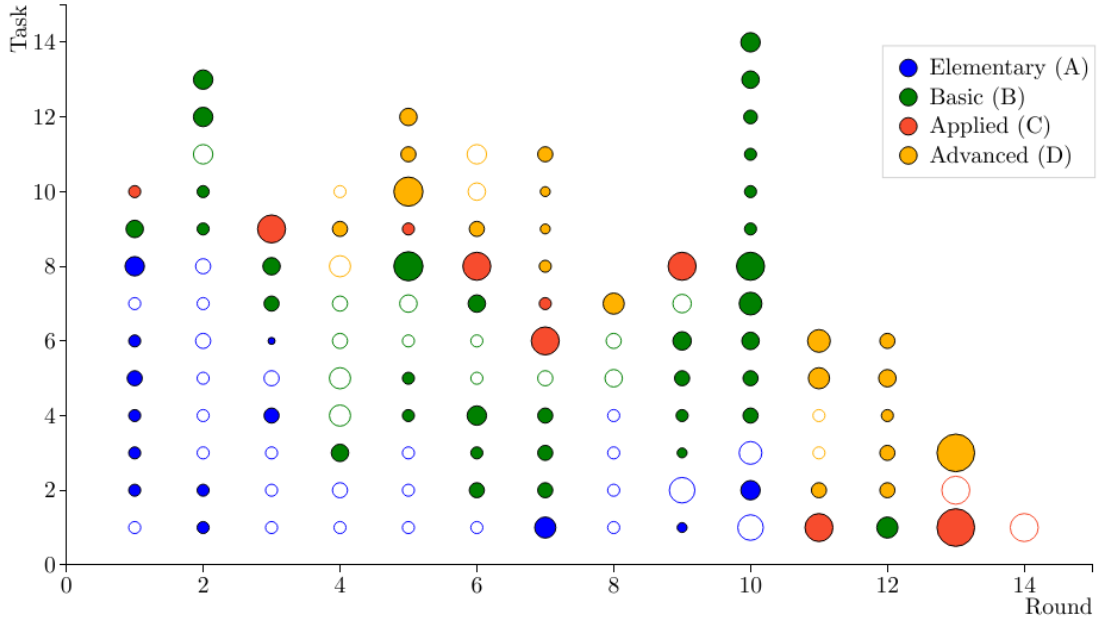

Fig. 6. Opportunist. Student who targeted grade 5 and achieved it. He had previous programming experience before this course and thus took advantage of the grading rules. He has only checked some of the A-level assignments but mainly skipped them.

Group 3: Casual well-performers: As can be perceived in Figure 4, a number of students targeting grades 3 and 4 exceeded their personal targets and ended up with a better grade. The course completion path visualizations of these students do not differ much from the visualizations of groups 1 and 2. We identified these students by also looking at the targets they had set for themselves on the third week of the course.

These students probably did not know exactly what to expect from this course. They just started completing the assignments and kept on working even if they had reached the point requirements of the grade they were targeting. The aim of the grading system was that you can change your targets during the course and this group of students proves that it works.

Group 4: Dropouts by missing assignments: Some students missed so many assignments that it was not possible to compensate any more. They gave up with the course and stopped working on the programming assignments. Typically, these students collected more points in the beginning of the course but then suddenly changed the direction completely.

There can be various reasons for this behavior. Some of the students realized that completing the course required more time than they had expected and allocated in their weekly schedules. Some of the students realized that they have already missed so many points that it was impossible to gain enough points by completing all the remaining assignments.

Group 5: Dropouts by missing skills: These students completed a lot of programming assignments and gathered enough points to pass the course. However, they failed due to not passing the exam. As explained earlier, the exam was not a difficult programming assignment, but similar to the regular assignments done during the course. What made the exam different from the regular assignments was that the student was expected to demonstrate the ability to write some code on their own in a controlled environment.

We have observed that there are students belonging in this group for two reasons: some of them co-operated all the time with a friend or a group of friends and some of them were regularly working in the multipurpose classroom waiting for the teaching assistant to solve all the problems they ran into. Nevertheless, the problem was the same for all of them: they did not learn the necessary programming skills to be able to work on their own and thus complete the programming assignment in the exam.

Group 6: Dropouts by failing personal expectations: These students were aiming at a high grade but collected enough points for passing at a lower grade only. They never took the exam of the course and thus failed the course. It seems that they are ambitious and did not want to have a bad grade in their study register. These students most likely signed up for the course's next implementation.

Group 7: Goal-driven students: In the beginning of the course, these students completed as many assignments as possible. When they gained enough points for one of the categories, they immediately stopped working on assignments in that category. The students aiming at grade 1 could gain enough points for passing the course at around week 9. Figure 7 illustrates a course completion path for a student in this group who stopped working on the course assignments after week 9. Figure 8 illustrates a rather more complicated course completion path from a student representing this group. He stopped working on the assignments in different categories step by step.

The teaching assistants reported that some students were complaining that the assignments in the higher levels were too difficult to be completed without completing A-assignments related to the same topic but there was no need to complete the A-assignments after fulfilling the A-point requirement. The complaining students were from this group and targeting grades 3-5. The complaints really encapsulate the aim of completing the programming assignments only to gain enough points. They did not see that completing the assignment in high level categories would be easier if you had first completed the easy assignments related to the same topic.

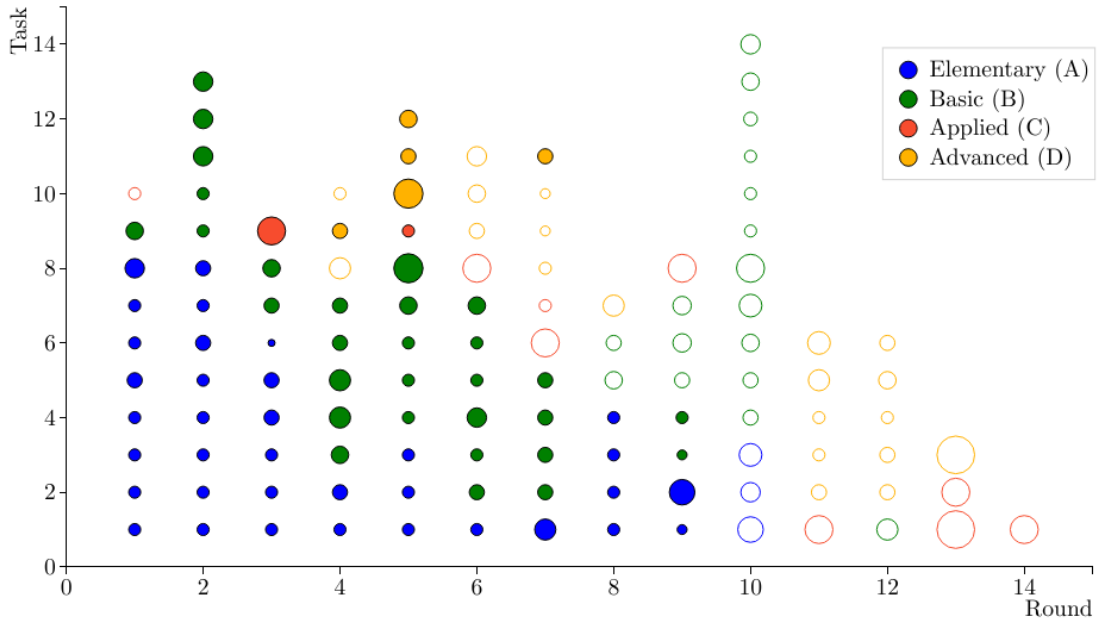

Fig. 7. Goal-driven. Student who targeted grade 1 and achieved it. In the beginning of the course he completed almost all the assignments. Then, on week 9 he gained enough of points for passing the course and stopped working on the course.

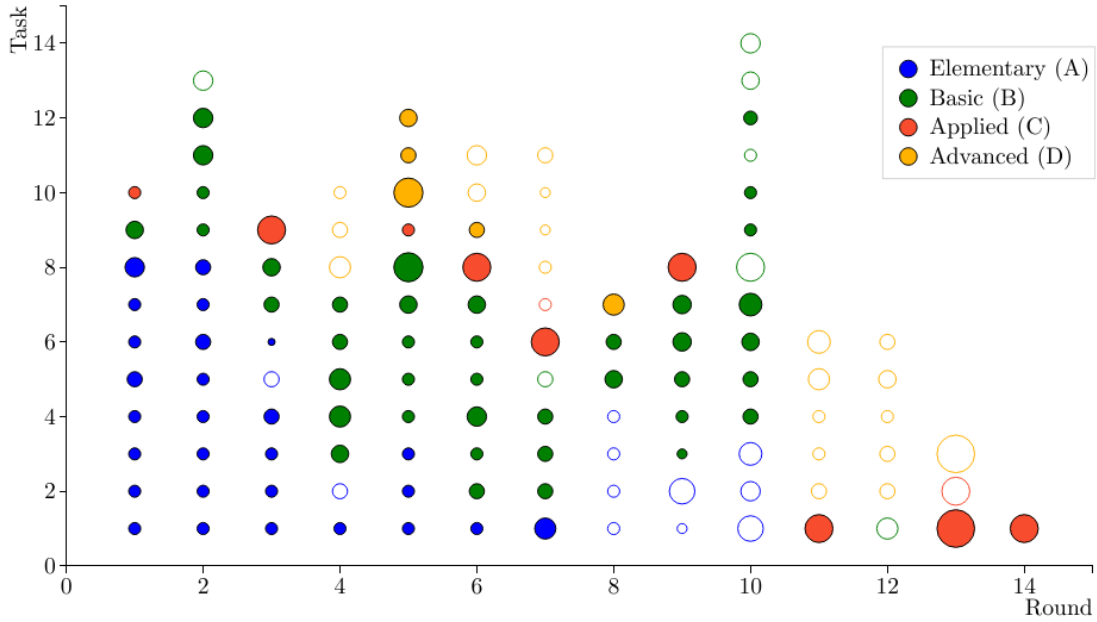

Fig. 8. Goal-driven. Student who targeted grade 3 and achieved it. He reached the requirements for A-points in week 7. After that he did not complete any A-assignments. Then, he reached the requirement for B-points in the middle of week 10, and did not complete any more B-assignments after that.

Group 8: Compensating students: These students had a phase of the course where they missed a lot of deadlines or handed in a lot of solutions by the extended deadline and thus missed points. For many students such an incident lead to dropping out from the course. However, the students in this group did not give up, but patched up the missing points by extra assignments in the end of the course. Figure 9 illustrates one course completion path of a student who was missing quite a lot of assignments in the first half of the course but did not dropout.

All the topics handled in the course build on the topics from the earlier weeks. This means that a student cannot just start working on the new assignment after missing all the assignments from previous week(s). You always need some knowledge from the earlier weeks to complete the new assignments. Therefore, patching up the missed points can become very laborious.

The aim of the grading was that none of the assignments is so important that missing it would lead to failing the course. This grading system allows the students to compensate missed assignments very liberally. However, compensating is not easy so the students really pay for the free time they have taken in the middle of the course.

A student with less thorough knowledge is most likely not able to compensate missing many assignments. We also observed multiple ways of dropping out from the course.

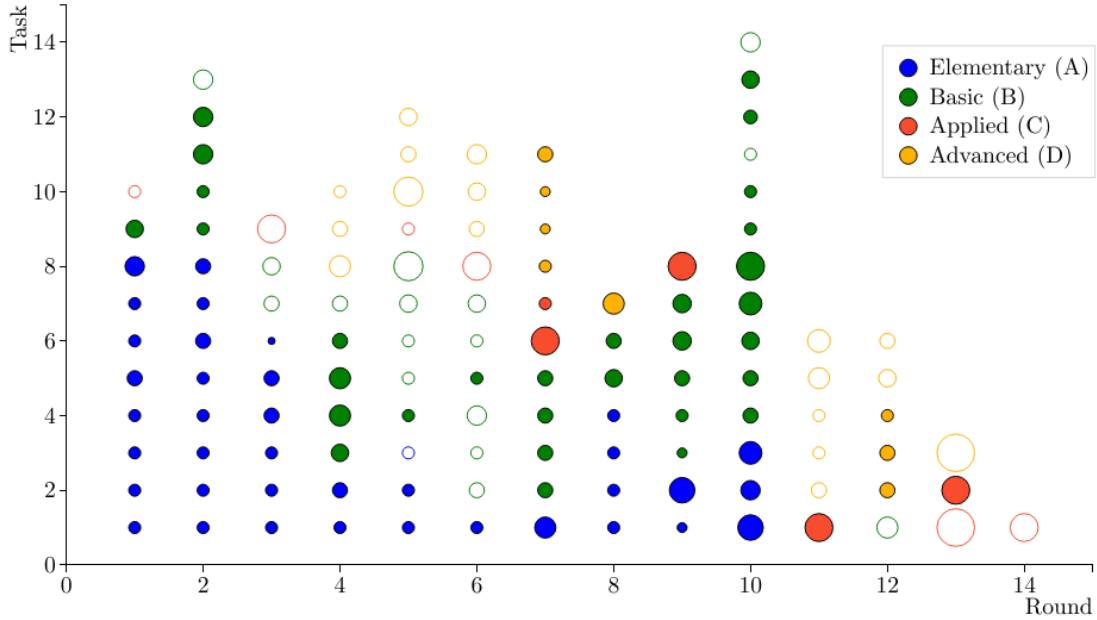

Fig. 9. Compensating. Student who targeted grade 3. However, he missed so many assignments that he only got grade 1. The biggest losses were the most difficult assignments in weeks 3-4 and almost all the assignments during weeks 5-6. Looking at the visualization quickly, it might seem that he has completed a lot of assignments. How- ever, there are many large red circles (weeks 3, 5, and 6) that are not colored. He was grouped in the compensating students, because he was able to catch up by completing some of the last assignments that were targeted for students targeting grades 3-5.

4.3 Student Feedback

This student-centered, agile style of arranging the course that allowed the students to work according to their own schedule was praised by the students who completed the course. Approximately 80% of students in each course implementation reported they worked “mostly independently” and over 10% “more often independently than in the multipurpose classroom”, when asked for feedback after the course was over. The open-ended feedback from students mentioned often, e.g. “It is very effective to be able to work on one’s own pace”. Of course, there was also negative feedback from students who did not like the approach of working independently but most of the feedback was positive.

The students who were not majoring in computer science also gave positive feedback regarding the possibility to pass the course with less work. However, some students criticized the grading, because they felt that it was unfair that you cannot have a good grade without completing the advanced assignments.

In the feedback over half of students checked that the course was too laborious. This, of course, was not our intention – rather, our idea was that the students work every week until their time for this course is up and then leave the rest of the assignments uncompleted. However, many of the students took more time for this course and completed most the assignments even if they did not manage to do it in the time they were planning to. The teaching assistants reported that many students felt that they need to have grade 5 because they can decide the grade themselves. In the traditional way of grading you typically do your best and hand in your solution for grading. Then the grader decides about your grade. Our way of grading turned this scene in the opposite direction: the students were able to decide their grade themselves by working for a longer time. Psychologically, there is a big difference in getting the grade that the grader decides for you and in making the decision yourself. For a student targeting grade 5, the decision to complete almost all the assignments had to be taken weekly again and again.

5 Discussion

Fundamentally, our goal was to be able to meet the needs of different student groups. The results section gave us insights into student feedback, the grades they achieved and different kinds of strategies of taking the course. While the feedback was generally positive and the students mostly liked the agile course setup, we also identified subjects for discussion and improvement.

5.1 Criterion referenced grading with automated feedback

In the "traditional" way of grading, it is possible to define the requirements for each grade after all submissions have been received. In the criterion referenced approach, the requirements have to be defined beforehand because they are the tool for guiding what students do. When this is combined with automated feedback, students know all the time what grades they have already earned and what grades are still reachable – provided they are able to pass the exam. After all, feedback students get from the assignments is one of the powerful influences on learning and achievement – either good or bad (Hattie & Timperley, 2007). In this study we observed both sides of feedback.

Automated assessment is sometimes criticized as it allows students to gather points from partially correct implementations. This can lead to a mismatch between actual and expected skills. In general, this could be tackled, for example, with a more coarse-grained grading of the assignments or by having some mandatory assignments where the feedback – even if automated – would be delayed (Spacco et al., 2006). Other approaches to address trial and error learning in the context of automatically assessed programming exercises are surveyed by Ihantola et al. (Ihantola, Ahoniemi, Karavirta, & Seppälä, 2010). We approached this challenge by grading nearly all assignments as passed/failed.

5.2 (Un)selecting the Assignments

When looking at the strategies defined in the previous section, both *Opportunists* and *Goal-driven students* followed a clear strategy for optimizing their workload. This anticipated pattern is demonstrated by advantage taking students skipping the easy assignments. Goal-driven students, however, made a sudden stop before the end of the course when they had enough points for their target grade. They might have been able to pursue a higher grade but as the criterion-based grading scheme was publicly available, they decided to use the remaining time for something else – hopefully for other courses they felt more relevant for their further studies.

An alternative interpretation for the *goal-driven group* could be that after a certain point, the assignments simply became too difficult. The fact that some goal-driven students applied also the opportunist strategy at the same time, and that some of the stops were between grades supports this. Thus, we can ask, is it reasonable to allow students to choose which assignments are valuable for their learning? If the grading rules direct the student in the wrong direction, correcting the course of learning takes a lot of self-discipline and good self-regulatory skills.

Moreover, looking at the groups *Dropouts by missing assignments* and *Compensating students*, we perceive that there is a very fine line between passing and failing the course. This raises the question, is it fair for students to allow them to skip assignments if the consequences are troublesome: compensating work or failing the course entirely. We assume that the difference between *Compensating students* and *Dropouts by missing assignments* is closely linked to the self-regulatory skills of students. Thus, in the future, we should support such skills as well. For example, gamification and various visualizations affect self-regulation (Auvinen, 2015). In addition, peer review can boost self-efficacy (Zingaro, 2014), which contributes to the students' persistence to pass the course.

5.3 Learning the Necessary Skills

The group *Dropouts by missing skills* did well during the course but failed the exam. We assume that free riding in pair programming could have led to disparity between the exam and the assignments. Thus, we argue that Dropouts by missing skills could be helped by adding a mid-term exam into the course requirements. This way they would recognize earlier that their way of working has led into problems with learning. If they recognized this early enough, it would still be possible to correct the problem before the final exam. Moreover, our electronic exam setting (Laine, Sipila, Anderson, & Sydänheimo, 2016) allows students to take the exam in their own schedule, so it would be easy to add the mid-term exam. We will consider it in the following course implementations. Another problem in our exam setup was that the students self-selected the difficulty of the exam. A longer exam where students would solve both easier and more demanding tasks should solve this problem.

5.4 The Requirements of Passing the Course

Finally, the group Dropouts by failing personal expectations is problematic from the teacher's perspective. The obligatory exam leaves students the possibility to decide to fail the course on purpose.

In our context, students can re-take this course and raise their grades later. The teachers of course wish that students would first make sure they pass and then later promote their grade if they still feel it is important. Thus, it seems that there is a need for redesigning the passing requirements so that it is not possible for the student to decide that he/she wants to fail the course like this. However, making the exam optional for grade 1 is not ideal, since the existence of the group Dropouts by missing skills proves that there is a need for a controlled exam. Midterm exam, discussed in the previous Section, and improved (automated) detection of extensive collaboration (Hellas, Leinonen, & Ihantola, 2017; Yan, McKeown, Sahami, & Piech, 2018) are the alternatives we consider to address these challenges in the future.

5.5 Over performing

In our study, the group Perfectionists did extra on the A-level assignments as well as over-performed by completing assignments where they learned new and exciting things (level C and D in our case).

There can of course be explanations to why students want to complete the voluntary A-assignments: Some of them do not have prior programming experience and thus need to learn all the details. The A-assignments are designed for this and thus useful. It is also possible that some students completed them in the beginning of the course just to do something since there were not many C- and D-level assignments at this phase of the course. However, in the later part of the course there were also A-assignments for the students who had difficulties in the course and thus missed some assignments here and there. These A-assignments are repeating the basic concepts that have already been learned in the earlier assignments. Therefore, we assume that a student who is going to have grade 5 will not learn much by completing these repetitive extra A-assignments. To our understanding, the only motivation of this student group for completing these repetitive extra A-assignments is that they want to have the maximum points on the course platform. Thus, we decided to name them the Perfectionists.

5.6 Summarizing Different Student Strategies

The course completion strategies resemble the strategies identified by Karavirta et al. (Karavirta, Korhonen, & Malmi, 2005). Based on the clustering of how students use resubmits in the context of automated assessment, they divide learners between: passers, ordinaries, iterators, ambitious and talented. One can argue that our Perfectionists are also ambitious. Advantage taking students are most likely the talented, and both compensating and goal driven students could be- long into passers. In the future, it would be extremely interesting to combine the information from resubmissions to learning paths and use the improved profiling for adaptive learning (Brusilovsky et al., 1998).

5.7 Further Observations

Finally, it can be regarded as unfair that the students, who fail to complete advanced assignments, cannot get grade 5. However, we believe it is also important to learn to set priorities, which is an important work-life skill for the later career – if you are not majoring in CS do you need to have the best grade in a laborious programming course? Granted, for a more homogeneous student group, such considerations might be misleading, but based on experience, students in any case set different priorities to their courses, and hence offering an agile, well-defined approach to do so has been regarded welcome.

6 Conclusions

In this chapter, we have provided an insight to the way that we provide personalized learning experiences in a university-wide introductory programming course. Course content is flexibly defined by the learners, and, based on the individual goal setting, the students are allowed to choose the assignments they work on. In analogy to agile software development, this corresponds to allowing the development team to decide which features they choose to work on, depending on the complexity and other factors associated with the feature. We have also shed light on how students experience and utilize the agile course setup by analyzing their study paths. In this respect, the following research questions directed our study.

1. How are students' self-reported target grades in the beginning of the course related to the final learning outcomes?

Just setting a goal – in our case, the target grade – seems to have a strong positive correlation with passing the course. Moreover, there is small/medium correlation between the

target grade and final grade. Interestingly, the means of the target grades in Autumn 2016 (including CS students) and Spring 2017 (mainly other students) were nearly the same, while the actual performance in Autumn 2016 was much better (see histograms in Figure 4). This hints that the presumably weaker students did not utilize the grading scheme as we expected. Perhaps it is difficult to set yourself low expectations. Given the way that the course was designed, this may have guided weaker students to do tasks that were too difficult.

2. What kinds of behavioral patterns can be detected among students in the agile course setup?

The main objective of this study was to understand how students utilize the agile course and how it works from the perspective of the diverse student population. However, in the process, we identified three distinct behavioral pat- terns that were related to high performance (i.e., Perfectionists, Opportunists, and Casual well-performers), three patterns of failing the course (i.e., Dropouts by missing assignments, by missing skills, and by failing personal expectations), and two other behavioral patterns associated with how students work (i.e., Goal- driven students and Compensating students).

The analysis of the goals and the perceived behavioral patterns show that there are various ways of completing the course. Unfortunately, we also perceived various ways of failing the course which suggests that the low performing students should be taken more into consideration when developing the course. Notwithstanding, the diversity of the students' strategies demonstrates that the criterion referenced grading scheme certainly supports the diversity of the student population better than a traditional course setup where all students complete the course in the same manner. Adjustment of the grading criterion is needed though.

Obviously, there are numerous directions for future work. In fact, almost any of the identified student groups could be more elaborately studied to understand their motives and goals. In particular, understanding why some talented and clearly capable students also complete simpler assignments rather than directly focusing on more complex problems is an interesting issue we plan to address in the future. In addition, studying how the students advance in their studies later on, and whether this correlates with their results and motivations in this course forms an interesting piece of further research.

References

Ahadi, A., Lister, R., Haapala, H., & Vihavainen, A. (2015, July). Exploring machine learning methods to automatically identify students in need of assistance. In *Proceedings of the eleventh annual International Conference on International Computing Education Research* (pp. 121-130). ACM.

Auvinen, T. (2015). Educational technologies for supporting self-regulated learning in online learning environments (Doctoral dissertation) Retrieved from http://urn.fi/URN: ISBN:978-952-60-6281-5

Biggs, J., & Tang, C. (2007). Teaching for Quality Learning at University 3rd edition. Open university Press.

Bishop, J. L., & Verleger, M. A. (2013, June). The flipped classroom: A survey of the research. In *ASEE National Conference Proceedings, Atlanta, GA, USA* (Vol. 30, No. 9, pp. 1-18).

Bloom, B. S., Engelhart, M. D., Furst, E. J., Hill, W. H., Krathwohl, D. R., et al. (1956). Taxonomy of educational objectives: The classification of educational goals. handbook I: Cognitive domain. New York: David McKay company. Inc. (7th Edition 1972).

Brusilovsky, P., et al. (1998). Adaptive educational systems on the world- wide-web: A review of available technologies. In *Proceedings of workshop" www-based tutoring" at 4th international conference on intelligent tutoring systems (ITS'98),* San Antonio, TX, USA.

Carter, J., Ala-Mutka, K., Fuller, U., Dick, M., English, J., Fone, W., & Sheard, J. (2003). How shall we assess this? In *ACM SIGCSE Bulletin* (Vol. 35, pp. 107–123).

Carter, J., Bouvier, D., Cardell-Oliver, R., Hamilton, M., Kurkovsky, S., Markham, S., . . . others (2011). Motivating all our students? In *Proceedings of the 16th annual conference reports on innovation and technology in computer science education-working group reports (ITiCSE'11).* (pp. 1–18). ACM.

Carter, J., White, S., Fraser, K., Kurkovsky, S., McCreesh, C., & Wieck, M. (2010). ITiCSE 2010 working group report motivating our top students. In *Proceedings of the 2010 ITiCSE working group reports* (pp. 29–47). ACM

Douce, C., Livingstone, D., & Orwell, J. (2005). Automatic test-based assessment of programming: A review. *Journal on Educational Resources in Computing (JERIC)*, 5 (3), 4.

Fuller, U., Johnson, C. G., Ahoniemi, T., Cukierman, D., Hernan-Losada, I., Jackova, J., . . . Thompson, E. (2007, December). Developing a computer science-specific learning taxonomy. SIGCSE Bulletin, 39 (4), 152–170.

Hattie, J., & Timperley, H. (2007). The power of feedback. Review of educational research, 77 (1), 81–112.

Hellas, A., Leinonen, J., & Ihantola, P. (2017). Plagiarism in take-home exams: Help-seeking, collaboration, and systematic cheating. In *Proceedings of the 2017 ACM conference on innovation and technology in computer science education* (pp. 238–243). ACM

Ihantola, P., Ahoniemi, T., Karavirta, V., & Seppälä, O. (2010). Review of recent systems for automatic assessment of programming assignments. In *Proceedings of the 10th Koli calling international conference on computing education research* (pp. 86–93).

Illeris, K. (2002). The three dimensions of learning. Malabar, Florida: Krieger Publishing Company.

Johnson, C. G., & Fuller, U. (2006). Is bloom's taxonomy appropriate for computer science? In *Proceedings of the 6th baltic sea conference on computing education research: Koli calling 2006* (pp. 120–123).

Johnson, G., Gaspar, A., Boyer, N., Bennett, C., & Armitage, W. (2012). Applying the revised Bloom's taxonomy of the cognitive domain to Linux system administration assessments. *Journal of Computing Sciences in Colleges*, 28 (2), 238–247.

Karavirta, V., Ihantola, P., & Koskinen, T. (2013, July). Service-oriented approach to improve interoperability of e-learning systems. In *Advanced Learning Technologies (ICALT), 2013 IEEE 13th International Conference on* (pp. 341-345). IEEE.

Karavirta, V., Korhonen, A., & Malmi, L. (2005). Different learners need different resubmission policies in automatic assessment systems. In *Proceedings of the 5th annual finnish/baltic sea conference on computer science education* (pp. 95–102).

Keim, D. A. (2002). Information visualization and visual data mining. *IEEE transactions on Visualization and Computer Graphics,* 8 (1), 1–8.

Laine, K., Sipila, E., Anderson, M., & Sydänheimo, L. (2016, 9). Electronic exam in electronics studies. In *SEFI annual conference* 2016.

Lister, R., & Leaney, J. (2003). Introductory programming, criterion-referencing, and bloom. *ACM SIGCSE Bulletin*, 35 (1), 143–147.

Mazza, R., & Milani, C. (2005). Exploring usage analysis in learning systems: Gaining insights from visualisations. *In Workshop on usage analysis in learning systems at 12th international conference on artificial intelligence in education* (pp. 65–72).

Merriam, S. (2001). Andragogy and self-directed learning: Pillars of adult learning theory. New Directions for Adult and Continuing Education (89), 3–14.

Romero, C., & Ventura, S. (2010). Educational data mining: a review of the state of the art. *IEEE Transactions on Systems, Man, and Cybernetics, Part C* (Applications and Reviews), 40 (6), 601–618.

Seddon, G. M. (1978). The properties of bloom's taxonomy of educational objectives for the cognitive domain. *Review of Educational Research,* 48 (2), 303–323.

Spacco, J., Hovemeyer, D., Pugh, W., Emad, F., Hollingsworth, J. K., & Padua- Perez, N. (2006). Experiences with Marmoset: designing and using an advanced submission and testing system for programming courses. *ACM Sigcse Bulletin,* 38 (3), 13–17.

Thompson, E., Luxton-Reilly, A., Whalley, J. L., Hu, M., & Robbins, P. (2008). Bloom's taxonomy for CS assessment. In *Proceedings of the tenth conference on australasian computing education conference.* volume 78 (pp. 155–161).

Yan, L., McKeown, N., Sahami, M., & Piech, C. (2018). Tmoss: Using intermediate assignment work to understand excessive collaboration in large classes. In *Proceedings of the 49th ACM technical symposium on computer science education* (pp. 110–115).

Zingaro, D. (2014). Peer instruction contributes to self-efficacy in CS1. In *Proceedings of the 45th ACM technical symposium on computer science education* (pp. 373–378).